

# ChatApp API Specification Document

Team Freedom

November 15, 2018

## 1 Overview

The ChatApp API consists of two packages: `controller`, `model`. Package `controller` communicates with the model and the view. Package `model` includes two packages `cmd`, `obj` and the `DispatchAdapter` file. Package `cmd` controls the effects when user send a request to back-end. Package `obj` contains user class, chat room class and message class. `DispatchAdapter` file communicates with the model and the view by passing back JSON data of our objects.

## 2 APIs

### 2.1 Package controller

#### 2.1.1 Class ChatAppController

The class that communicates with all the clients on the web socket. It is the entry of the Chat App. It starts the web socket for the server and specifies the endpoint for the web socket.

#### 2.1.2 Class WebSocketController

The class that represents the web socket for the server. It uses a dispatcher adapter to dispatch different events from web sockets. Three class methods are listed as follows.

- `public void onConnect(Session User)`  
Defines the action when a new user session is connected.
- `public void onClose(Session user, int statusCode, String reason)`  
Defines the action when a user session is closed.
- `public void onMessage(Session user, String message)`  
Defines the action when a user session sends messages to the server.

### 2.2 Package model.cmd

In the Chat App, we use Command Design Pattern. Every event will be parsed into a command and then the command will be passed next for according objects to execute.

### 2.2.1 Interface IUserCmd (Command)

Interface for all kinds of commands. All concrete classes that implement `IUserCmd` need to have method `execute(User context)`, which executes a command on a user to update its attributes.

### 2.2.2 Class AppendRoomCmd (Command, Singleton)

Class that defines the command when a new chat room is created. It implements `IUserCmd`. Attributes of the class are listed as follows.

- `ChatRoom room` the new created chat room.

One class method is listed as follows.

- `public void execute(User context)`  
Defines the command when a new chat room is created. Its parameter should be all users. The context user will have this new created chat room as a available chat room.

### 2.2.3 Class CollectNamesCmd (Command, Singleton)

Class that defines the command when a chat room's user list updates and need to get updated users' ids and names. It implements `IUserCmd`. Attributes of the class are listed as follows.

- `Map<Integer, String> names` the user-id, user-name map.

One class method is listed as follows.

- `public void execute(User context)`  
Defines the command when a chat room's user list updates and need to get updated users' ids and names. Its parameter should be users in the updated chat room. The map will add the context user's id and name.

### 2.2.4 Class DeleteRoomCmd (Command, Singleton)

Class that defines the command when a chat room is deleted by its owner. It implements `IUserCmd`. Attributes of the class are listed as follows.

- `ChatRoom room` the deleted chat room.

One class method is listed as follows.

- `public void execute(User context)`  
Defines the command when a chat room is deleted by its owner. Its parameter should be all users. The context user will delete the chat room from his/her joined and available chat room lists.

### 2.2.5 Class EnforceFilterCmd (Command, Singleton)

Class that defines the command when qualification of a chat room is reseted by its owner. It implements IUserCmd. Attributes of the class are listed as follows.

- ChatRoom room the chat room which resets its qualifications.

One class method is listed as follows.

- public void execute(User context)  
Defines the command when qualification of a chat room is reseted by its owner. Its parameter should be users in the chat room. If the context user does not satisfy the new qualification, he/she will removed from the chat room.

### 2.2.6 Class NotifyClientCmd (Command, Singleton)

Class that defines the command when a chat room changes, all it users should also update themselves. It implements IUserCmd. Attributes of the class are listed as follows.

- Map<String, String> info; the update info.
- DispatcherAdapter dis; the dispatcher adapter who deals with update info.

One class method is listed as follows.

- public void execute(User context)  
Defines the command when an chat room changes, all it users should also update themselves. Its parameter should be all users of the chat room. The dispatcher adapter will update the context user according to the update info.

## 2.3 Package model.obj

### 2.3.1 Class User (Observer)

Class User initializes a new user with required attributes like: session, name, age, joined chat room, etc. It also has methods to chat with other users or create/join/leave chat room. Class User has following private fields:

- int id is the id of user.
- Session session is the session of user.
- String name is the name of user.
- int age is the age of user.
- String location is the location of user.
- String school is the school of user.
- String name is the name of user.
- List<Chatroom> joined is the joined chat rooms of user.

- `List<Chatroom> available` is the available chat rooms of user.

All private fields are initialized when a new user is created. Class methods are listed as follows:

- `public int getId()`  
Return the id of user.
- `public Session getSession()`  
Return the session of user.
- `public String getName()`  
Return the name of user.
- `public int getAge()`  
Return the age of user.
- `public String getLocation()`  
Return the location of user.
- `public String getSchool()`  
Return the school of user.
- `public List<ChatRoom> getJoined()`  
Return the joined chat rooms of user.
- `public List<ChatRoom> getAvailable()`  
Return the available chat rooms of user.
- `void addRoom(ChatRoom room)`  
Add available chat room to user's available list.
- `void removeRoom(ChatRoom room)`  
Delete chat room from user's available and joined list.
- `void moveToJoined(ChatRoom room)`  
Move available chat room to joined list if user joins that room.
- `void moveToAvailable(ChatRoom room)`  
Move joined chat room to available list if user leaves that room.
- `boolean joinRoom(ChatRoom room)`  
Determine whether user can join the chat room, if yes, add user to that room and remove that room from user's available room list.
- `void boolean leaveRoom(ChatRoom room)`  
Remove user from the chat room.
- `void refresh(ChatRoom room)`  
Transfer current user's chat room info in JSON format to client via dispatch adapter and then update client according to user's info.

### 2.3.2 Class ChatRoom (Observable)

Class `ChatRoom` initializes a new chat room with required attributes like: age, owner, qualification, etc. It also has methods to add/remove users and provide interface to owner to control the chat room. Class `ChatRoom` has following private fields:

- `int id` is the id of chat room.
- `String name` is the name of chat room.
- `int ageLowerBound` is the lower bound age of chat room.
- `int ageUpperBound` is the upper bound age of chat room.
- `String[] locations` is the location requirement of chat room.
- `String[] schools` is the school requirement of chat room.
- `DispatcherAdapter dis` is the adapter to communicate with model and views.
- `List<String> notifications` is the notifications contain why the user left, etc.
- `Map<String, List<Message>> chatHistory`  
Maps key("smallIdlargeId") to list of chat history strings.

All private fields are initialized when a new chat room is created. Class methods are listed as follows:

- `public int getId()`  
Return the id of chat room.
- `public String getName()`  
Return the name of chat room.
- `public User getOwner()`  
Return the owner of chat room.
- `public List<String> getNotifications()`  
Return the notifications of chat room.
- `public DispatcherAdapter getDispatcher()`  
Return the dispatch adapter of chat room.
- `public Map<Integer, String> getUsers()`  
Return the id and name of user who joins the chat room.
- `public void modifyFilter(int lower, int upper, String[] locations, String[] schools)`  
Modify the qualification of the chat room.
- `public boolean applyFilter(User user)`  
Check if the user meets the requirement to join the chat room.
- `public void addUser(User user)`  
Add user to chat room.

- `public void removeUser(User user, String reason)`  
Remove user from chat room and send reason to all users in the chat room.
- `public void storeMessage(User sender, User receiver, Message message)`  
Store chat history of the chat room.
- `public void freeChatHistory(User user)`  
Parse the key and remove chat history related to user.
- `public void refresh()`  
Refresh the chat room to update notification and user list.

### 2.3.3 Class Message

Class `Message` initializes a new message with required attributes like: sender's id, receiver's id, message content, etc. It also has methods to get/set these info. Class `Message` has following private fields:

- `int id` is the id of message.
- `int roomId` is the room's id of message.
- `int senderId` is the sender's id of message.
- `int receiverId` is the receiver's id of message.
- `String message` is the content of message.
- `boolean isReceived` means whether this message is received by user.

All private fields are initialized when a new message is created. Class methods are listed as follows:

- `public int getId()`  
Return the id of message.
- `public int getRoomId()`  
Return the room's id of message.
- `public int getSenderId()`  
Return the sender's id of message.
- `public int getReceiverId()`  
Return the receiver's id of message.
- `public String getMessage()`  
Return the content of message.
- `public boolean isReceived()`  
Return the status of message.
- `public void setReceived()`  
Return the status of message.

## 2.4 Class DispatcherAdapter (Observable)

Communicates with the model and the view by passing back JSON data of objects. It is an observable and will add created objects as its observers. When state of model changes, the dispatcher adapter will notify the observers to update themselves. Attributes of class are listed as follows.

- `int nextUserId` is the id of new user.
- `int nextRoomId` is the id of new room.
- `int nextMessageId` is the id of new message.
- `Map<Integer, User> users`  
maps user id to the user.
- `Map<Integer, ChatRoom> rooms`  
maps room id to the chat room.
- `Map<Integer, Message> messages`  
maps message id to the message.
- `Map<Session, Integer> userIdFromSession`  
maps session to user id.

All private fields are initialized when a new message is created. Class methods are listed as follows:

- `public void newSession(Session session)`  
Create a new session and enlarge the id of user by 1.
- `public int getUserIdFromSession(Session session)`  
Return the id of user from corresponding session.
- `public User loadUser(Session session, String body)`  
Load a user into the environment and return that loaded user object.
- `public ChatRoom loadRoom(Session session, String body)`  
Load a chat room into the environment and return that loaded chat room object.
- `public void unloadUser(int userId)`  
Remove a user with given userId from the environment.
- `public void unloadRoom(int roomId)`  
Remove a room with given roomId from the environment.
- `public void joinRoom(Session session, String body)`  
Add a user to chat room, body is of format "roomId"
- `public void leaveRoom(Session session, String body)`  
Remove a user from chat room, body is of format "roomId"
- `public void modifyRoom(Session session, String body)`  
Modify the joining qualification of the chat room, body is of format "roomId + lower + upper + {[location],}\*{[location]} + {[school],}\*{[school]}"

- `public void sendMessage(Session session, String body)`  
A sender sends a string message to a receiver, body is of format "roomId + receiverId + rawMessage".
- `public void ackMessage(Session session, String body)`  
Acknowledge the message from the receiver, body is of format "msgId".
- `public void notifyClient(User receiver, Map<String, String> info)`  
Notify the client for refreshing.
- `public void query(Session session, String body)`  
Send query result from controller to front end, body is of format "type + roomId + [senderId] + [receiverId]".
- `public Map<Integer, String> getUsers(int roomId)`  
Return the names of all chat room members.
- `public List<String> getNotifications(int roomId)`  
Return notifications in the chat room.
- `public List<Message> getChatHistory(int roomId, int userAId, int userBId)`  
Return chat history between user A and user B (commutative).

## 3 Use Cases

### 3.1 Chat App Client

The whole chat room client is a board including three parts. The first left part consists of joined chat rooms, available chat rooms and a "+" symbol which is used to create a new chat room. The second middle part will show the owner and member in selected chat room, user can talk to a user by clicking the avatar. The third right part includes a board showing chat message and a input area where you can type and send message.

### 3.2 Objects

#### 3.2.1 ChatRoom

`ChatRoom` is a `Observable` object consisting of `Observer` objects `User`. There is two types of chat rooms: joined chat room and available chat rooms. The joined room means user is in that room, the available room means user is not in but can join that room.

#### 3.2.2 User

`User` is created by signing in. Each user is unique and but can belong to many chat room. If user create a chat room, he/she is the owner of that room therefore he/she can determine whether a user can join his/her room and send message to all users in that room.



### **3.2.3 Message**

Message is the information sent between users. It can be stored in chat room where the message is transmitted. It contains identity of sender and receiver as well the status of whether it is received by user.

## **3.3 Uses Description**

### **3.3.1 Login**

User can sign in Chat App via filling in his/her personal information including name, age, location, school. after signing in, user will have a unique account which is used to chat with other users and create/join/leave chat rooms.

### **3.3.2 Create chat room**

User can create a chat room by clicking "+" symbol at the left lower corner. After clicking, user should make the requirements of the new room including age, location, school and etc. A user can create any number of chat rooms. Creating a chat room means this user is the owner of that room so he can do the following operations:

- Chat with users  
User can chat with any user belonging to the same room through clicking the avatar listed in chat room board. After clicking, there will be a chat window in the right part of App in which message shows in the upper part and user types and sends message in the lower part.
- Send message to all users in the room Owner of chat room can send message to all users in the chat room.
- Change room requirement Owner of the chat room can change the requirements of chat room. For example, he can change the age range according to his willing. Any one who do not match the requirements can not join the room.

### **3.3.3 Join chat room**

Joining the chat room means he is not owner of the chat room. He can not send message to all users in the chat room and change the qualification of chat room.

- Chat with users User can chat with any user belonging to the same room through clicking the avatar listed in chat room board. After clicking, there will be a chat window in the right part of App in which message shows in the upper part and user types and sends message in the lower part.

### **3.3.4 Leave chat room**

User can choose to leave the chat room. After leaving, all users remaining would receive a notification saying "user xx leaves chat room because xxx". If owner of chat room leaves the chat room, the room would be destroyed.

## 4 Design Decisions

To simplify the chat app implementation, we made some design decisions, which are listed as follows.

### **Chat room is destroyed if owner leaves room**

If the owner of chat room leaves the room. the room would be automatically destroyed by client.

### **Name of user and chat room should be unique**

The name of user and chat room should not be the same with each other since we use the name to identify different user and room.

### **No room before creating/joining**

There is no room before creating or joining chat rooms on user's room board.

### **Self-defined API communication protocol**

Login

- Grammar: login [userName] [age] [location] [school]
- Example: login Shakeshack 30 USA Rice

Create a user

- Grammar: create [roomName] [ageLower] [ageUpper] {[location],}\*{[location]} {[school],}\*{[school]}
- Example: create Freedom 20 80 USA,Canada Rice,Harvard

Modify chat room requirements

- Grammar: modify [roomId] [ageLower] [ageUpper] {[location],}\*{[location]} {[school],}\*{[school]}
- Example: modify 0 30 50 USA, Harvard,MIT

Join a chat room

- Grammar: join [roomId]
- Example: join 1

Leave a chat room

- Grammar: leave [roomId]
- Example: leave 1

Send a message to receiver at a room

- Grammar: send [roomId] [receiverId] [message]

- Example: send 2 1 Hello, how are you?

Receiver ack a message from sender at chat room

- Grammar: ack [msgId]
- Example: ack 10

Query room user from chat room

- Grammar: query roomUsers [roomId]
- Example: query roomUsers 3

Query room notifications from chat room

- Grammar: query roomNotifications [roomId]
- Example: query roomNotifications 3

Query user chat history from chat room

- Grammar: query userChatHistory [roomId]
- Example: query userChatHistory 3

## 5 Summary

Our ChatApp API has four critical parts. Objects of user, chat room and message are defined in `obj` package. The `controller` package is the entry of App. The `cmd` package includes command strategies to execute specific order. The class `DispatchAdapter` observes the whole chat app system.